iRobot Create/CASA User Manual

Rob Kremer
Department of Computer Science
University of Calgary
2500 University Dr.
Calgary, Alberta, Canada, T2N 1N4
Email: kremer@cpsc.ucalgary.ca

February 21, 2013

Abstract

CASA (Collaborative Agent Systems Architecture) is a framework for writing collaborative agents. The iRobot/CASA system is build upon CASA allows control of iRobot Create /citep? robots using a bluetooth interface through the BAM (Element Products Inc., 2007) iRobot Create accessory. In addition, a simulation environment is supported.

Contents

1 Introduction

CASA (Collaborative Agent Systems Architecture) is a framework for writing collaborative agents. The iRobot/CASA system is build upon CASA allows control of iRobot Create (iRobot Inc., 2006a,b) robots using a bluetooth interface through the BAM (Element Products Inc., 2007) iRobot Create accessory. In addition, a simulation environment is supported.

CASA and the iRobot extension is written in Java 6.0 and uses Armed Bear Common Lisp (Common-Lisp.net, 2011) for ontology, conversation and policy declarations, as well as for scripting and run-time control (including commands to the robot). iRobot/CASA programmers may define agent behaviour in terms of either Java or Lisp code, and have that behaviour called upon by policies defined in either Java or Lisp code.

2 Getting Started

CASA and iRobot Create/CASA are written in Java, so can be run on any operating system supporting Java. CASA and iRobot Create/CASA require at least a Java 6.0 runtime.

The iRobot Create/CASA jar files can be downloaded from the CASA website at http://casa.cpsc.ucalgary.ca/iRobotCreate3 from the download page, http://casa.cpsc.ucalgary.ca/iRobotCreate3/dist/index.html.

2.1 Running iRobot Create/CASA from the command line

To run from the command line, use the following command line:

```
java -jar iRobotCreate.jar CASA-Lisp-command
```

where *CASA-Lisp-command* may be any Common Lisp command or function call understood by CASA. Most commonly, this command may be `(load` *`filename`*`)`, which will run the Lisp commands in *filename* as a script. Several examples of scripts for iRobot Create robots and the simulator can be found on the download page.

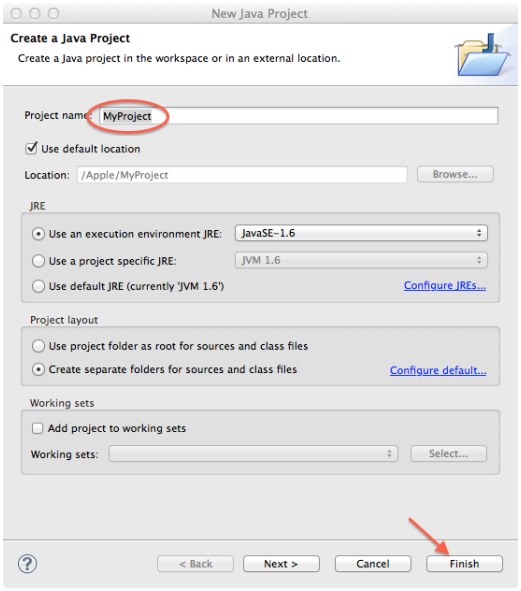
For an explanation of CASA Common Lisp commands, see the CASA User Manual (Kremer, 2012), §10 “Scripting language: Lisp”.

Additional Jars can be supplied using the Java command line `-classpath` (or `-cp`) qualifier.

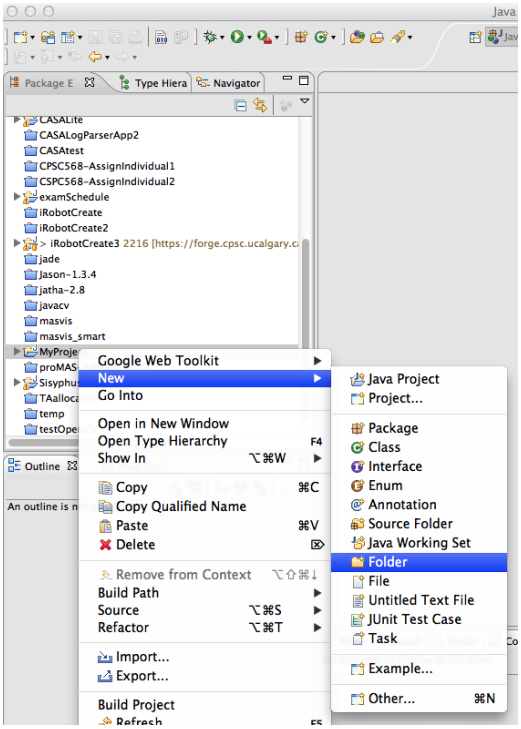
2.2 Loading Create/CASA as an Eclipse Project

1. Menu “`file` | `new` | `Java Project`”

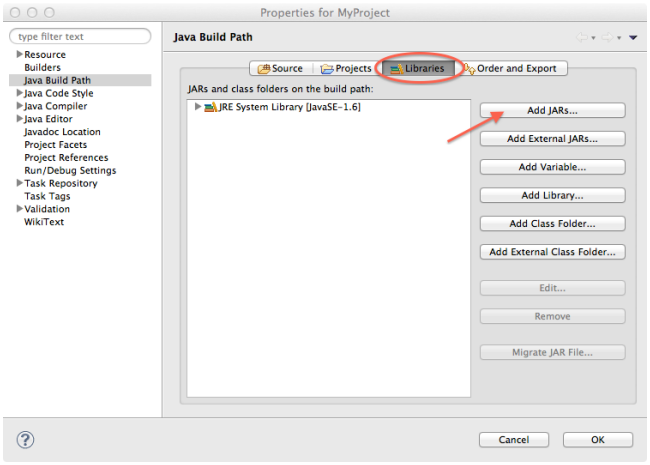
2. Fill out the “Project name” field, and choose “Finish”:



3. Create a new new folder in your new project, and call it “jars”:

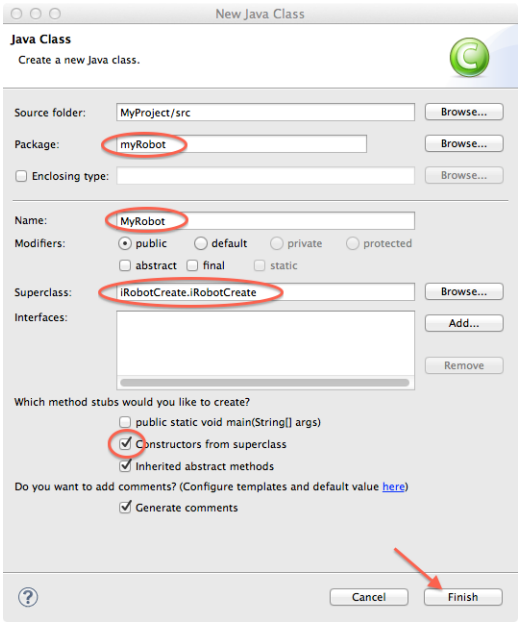


4. Move the iRobotCreate.jar file into the new *jars* folder.
5. Right-click on your project in Package Explorer to bring up the context menu. Choose "`Build Path` | `Configure Build Path...`".
6. In the "Properties for *project* | Java Build Path" dialog box click on the "Libraries" tab. Then choose "Add JARs...":

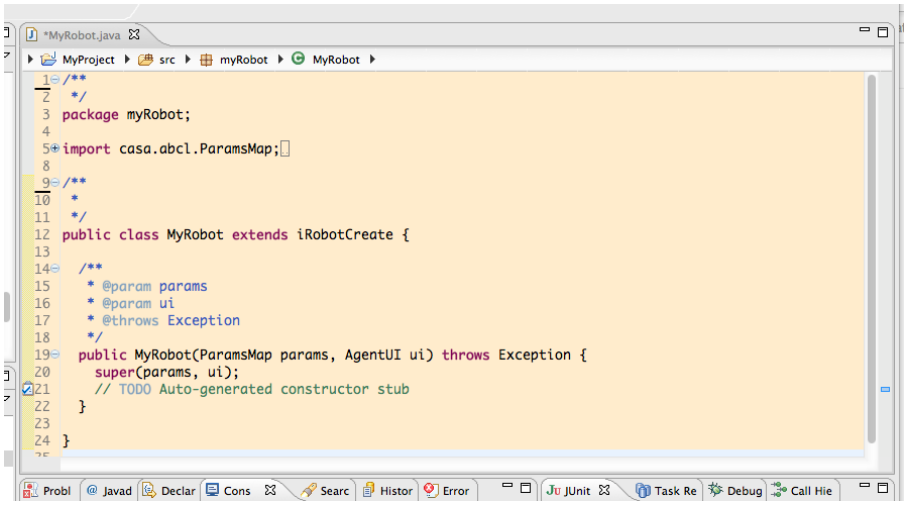


   then choose the iRobotCreate3.jar file from your .jars folder.
7. Hit "OK" in the "Properties for *project*" dialog box.
8. You are now ready to create your own iRobot agent.
9. Right-click on your project in Package Explorer to bring up the context menu. Choose "`New` | `Class`".

10. Fill in the following fields in the "New Java Class" dialog box, and click "Finish":



11. You should have a file like this:



This is a basic agent that inherits the capabilities of the iRobotCreate class, and will allow you control the robot from the command line in Lisp. You can also extend it to add new behaviours.

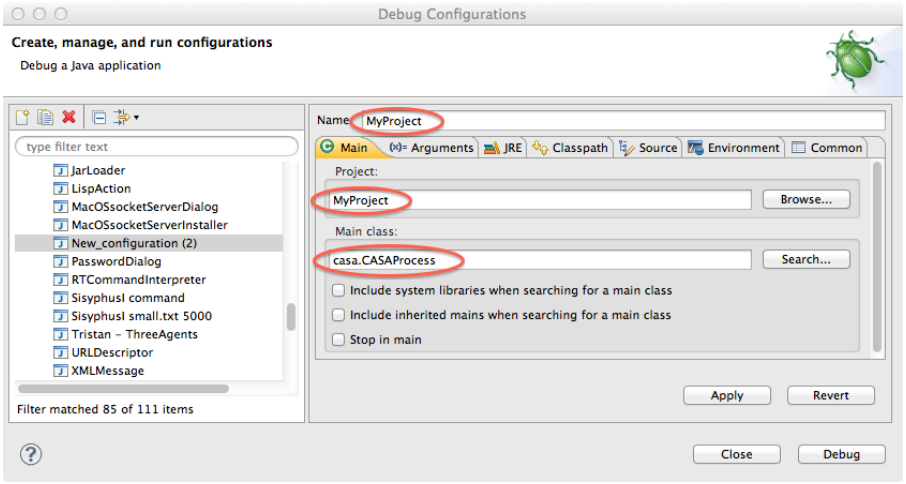
12. To demo your new robot class from Eclipse, you will need a script. We choose to run a the simulated environment using the following demo script:

```
(let (
       (trace-tags "info5,warning,msg,iRobot,-boundSymbols,-policies9,-commitments,-eventqueue,-conversations")
       (trace-code 10) ; bit 1=off, 2=on, 4=to-monitor-window, 8=trace-to-file
       (sleep-time 2) ; time to sleep between starting dependent agents, adjust for slower machines
       )

  ;; Set the options for the agent running the commandline
  (agent.options :options.tracing T)
  (agent.options :options.tracetags trace-tags)

  (sleep 2)

  (agent.new-agent "iRobotCreate.simulator.Environment" "RoomEnvironment" 5780 :LAC 9000
            :process "CURRENT" :trace trace-code :traceTags trace-tags :markup "KQML")
  (sleep 5)

  (agent.tell ":5780" "(iRobot-env.puck :name p̈uck)̈")
  (agent.tell ":5780" "(iRobot-env.set p̈uck¨ :labeled NIL)")
  (agent.tell ":5780" "(iRobot-env.circle p̈uck¨ :color-name r̈ed)̈")

  (sleep-ignoring-interrupts 2)
  (agent.new-agent "myRobot.MyRobot" "Alice2" 9100  :LAC 9000 :process "CURRENT" :trace trace-code
          :traceTags trace-tags :markup "KQML" :outstream "Alice.out" :instream "Alice.in")
  (agent.new-agent "myRobot.MyRobot" "Bob2"   9101  :LAC 9000 :process "CURRENT" :trace trace-code
          :traceTags trace-tags :markup "KQML" :outstream "Bob.out" :instream "Bob.in")
  ) ;let
```

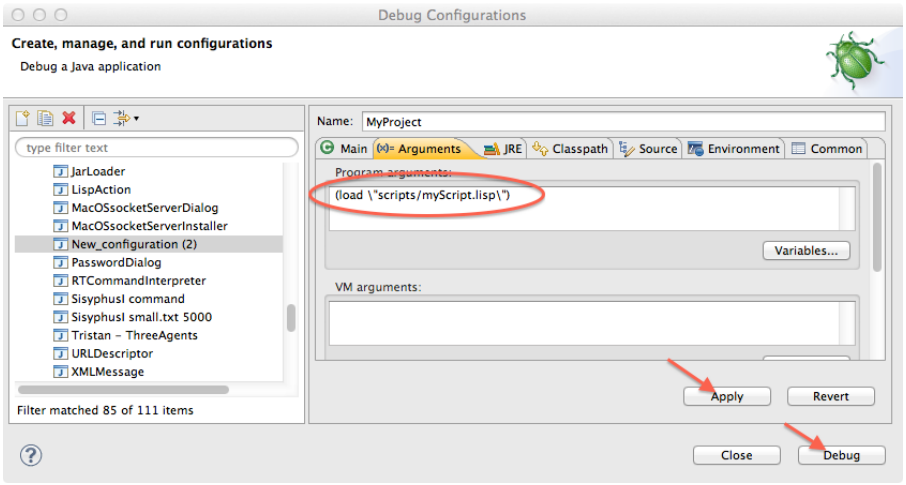
Place this file in your project in a subdirectory of your project file and call it “scripts”. Name the file “myScript.lisp”.

13. To run it in the debugger, left-click on downarrow (menu pulldown) of the Eclipse debug icon and choose “Debug Configurations...”.

14. Click on the "New" button ( ) and, in the "Main" tab, fill out the following fields:

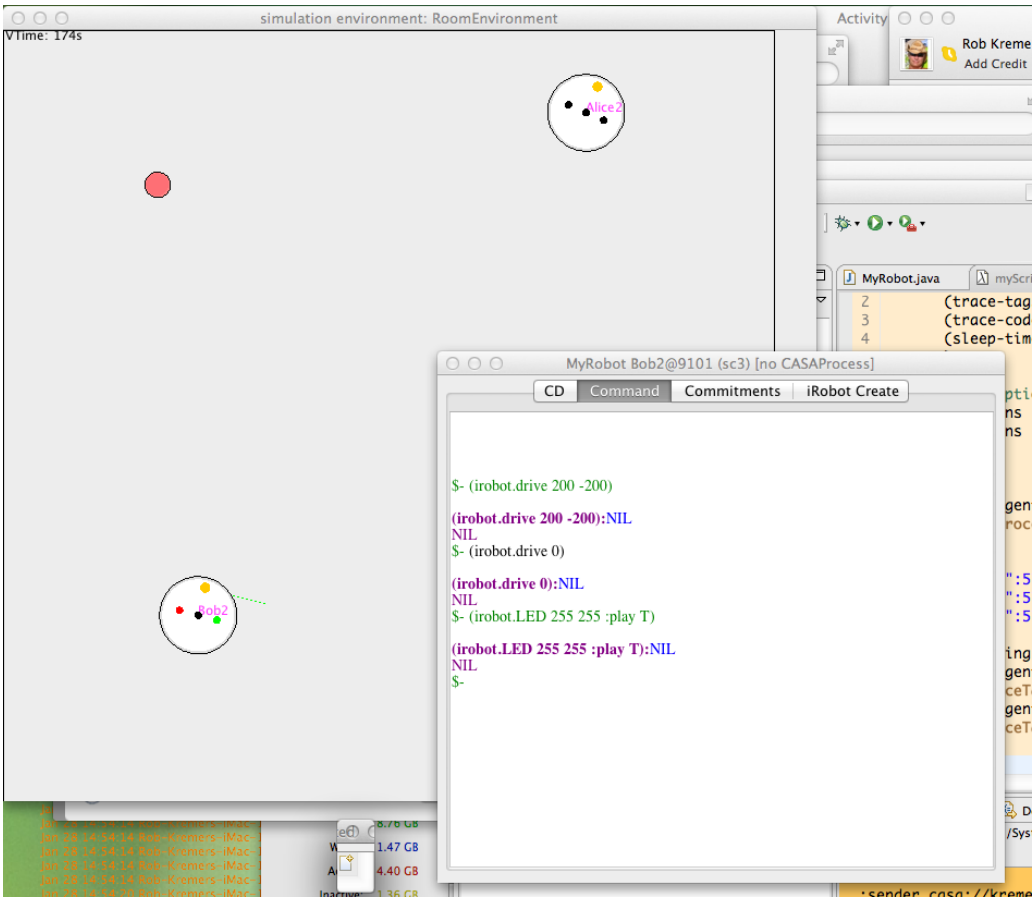


15. Click on the "Arguments" tab and fill out the "Program arguments" field:



Then click "Apply" and "Debug".

16. You should (perhaps after a considerable pause) get several windows showing up:



You can use the Robot agent windows' "Command" tabs to control the robots as demonstrated in the figure. Likewise, you can edit the environment in the Environment agent's "Command" tab. Depending on your configuration, you might just see the "Command" tab for a LAC or Process agent. If you do, click on the "Desktop" tab to see an MDI pane containing the various agent interface windows.

3 iRobot Create/CASA Commands

3.1 drive

Drives the robot forward: first changes mode to the SAFE state and then issues a DRIVE command. Negative velocities mean backward. A +ve velocity with a +ve radius means "drive forward while turning toward the left". A +ve velocity with a -ve radius means "drive forward while turning toward the right". Special cases:

- Straight: radius = 32768 or 32767 (0x8000 or 0x7FFF)

- Turn in place clockwise: -1 (0xFFF)
- Turn in place counterclockwise: +1 (0x0001)

The parameters are:

- `velocity` is in mm/second between -500 and 500.
- `radius` signifies the radius of a circle along whose circumference the robot follows with values between -2000 and 2000. Default: 0x7FFF (straight).
- `bypass` the command bypasses the command queue to be sent to the robot immediately. Default: false.
- `flush` the command queue is flushed empty (without being executed) and the command is sent to the robot immediately. Default: false.

Lisp

```
(irobot.drive velocity &optional (radius #7FFF) &key (bybass nil) (flush nil))
```

Java

```
protected void drive(short velocity)
protected void drive(short velocity, short radius)
protected void drive(short velocity, short radius, boolean bypassQ)
protected void drive(short velocity, short radius, boolean bypassQ, boolean flush)
```

3.2 move-by

Drives the robot in a straight line by approximately `distance` mm. Negative quantities move the robot backwards.

Lisp

```
(irobot.moveby distance)
```

Java

```
protected long moveBy(short mm)
```

3.3 reset

Attempts to reset the robot. (Usually fails.)

Lisp

```
(irobot.reset)
```

Java

```
public void resetRobot()
```

3.4 demo

Run the iRobot Create internal demo program number `number`.

Lisp

```
(irobot.demo number)
```

Java

```
public void doDemo(byte demoNumber)
```

3.5 dock

Attempt to dock the robot in the recharging station. This is actually `(demo 1)`.

Lisp

```
(irobot.dock)
```

Java

```
public void dock()
```

3.6 execute

Sends commands in arg to the robot. A command string in `arg-string` is a string of decimal bytes, but there are two modifiers 's' interprets the immediately following number as a 16-bit short (two bytes), and 'p' pauses the output fed to the robot for the milliseconds specified by immediately following number. For example, the argument value
`137 s-256 s257 p1000 137 s0 s0`
sends 137, 1, 0, 1, 1, 137, 0, 0, 0, 0 which drives the robot backwards in an arc for 1 second, and then stops it.

Lisp

```
(irobot.execute arg-string)
```

Java

```
public Status executeP(String arg-string)
```

3.7 execute-raw

Sends commands in `arg-string` to the robot. A command string in `arg-string` is a string of decimal bytes, but there is a modifiers 's' interprets the immediately following number as a 16-bit short (two bytes). For example, the argument value
`137 s-256 s257`
sends 137, 1, 0, 1, 1 which drives the robot backwards in an arc.

Lisp

```
(irobot.execute-raw arg-string)
```

Java

```
public Status executeRaw(String arg-string)
```

3.8 mode

Get/Change the robot's `mode` (0=Off | 1=Passive | 2=Safe | 3=Full). If
textttmode is omitted, the current mode is returned; if it's given, the mode is set.

Lisp

```
(irobot.mode &optional mode)
```

Java

```
protected int getMode()
protected void setMode(byte mode)
```

3.9 rotate-deg

Rotate the robot by an `angle` in degrees (-ve is clockwise).

Lisp

```
(irobot.rotate-deg angle)
```

Java

```
protected long rotateAngle(short angle)
```

3.10 rotate-mm

Rotate the robot by `distance` in mm (-ve is clockwise).

Lisp

```
(irobot.rotate-mm distance)
```

Java

```
protected long rotate(short distance)
```

3.11 LED

Set the LEDs on the robot.

- `colour` sets the color of the power LED (0=green to 255=red).
- `intensity` sets the intensity of the power LED (0=off to 255=full on).
- `play` sets the play button (NIL=off, other=on).
- `advance` sets the advance button (NIL=off, other=on)

Lisp

```
(irobot.LED &optional colour intensity &key play advance)
```

Java

```
protected int buttonColor
protected int buttonIntensity
protected boolean buttonPlay
protected boolean buttonAdvance
```

3.12 breathing

Tests if the robot is breathing (listening to commands – it might be busy executing a script). Normally, a test of breathing will be cautious, and delay; setting `peak` will not delay.

Lisp

```
(irobot.breathing &key peak)
```

Java

```
protected boolean breathing
protected boolean breathing()
```

4 iRobot Create/CASA sensors

The following predicate values are asserted into the agent knowledge base whenever the corresponding sensors are updated:

- BumpsAndWheelDrops
- Wall
- CliffLeft
- CliffFrontLeft
- CliffFrontRight
- CliffRight
- VirtualWall
- Overcurrents
- Unused1
- Unused2
- IRByte
- Buttons
- Distance
- Angle
- ChargingState
- Voltage
- Current
- BatteryTemperature
- BatteryCharge
- BatteryCapacity
- WallSignal
- CliffLeftSignal
- CliffFrontLeftSignal
- CliffFrontRightSignal
- CliffRightSignal
- UserDigitalInputs
- UserAnalogInput
- ChargingSourcesAvailable
- OIMode
- SongNumber
- SongPlaying
- NumberOfStreamPackets
- Velocity
- Radius
- RightVelocity
- LeftVelocity

4.1 Getting Sensor Values

The easiest way to access the values of the above predicates is using the Lisp command (for example):

```
(kb.get-value "Wall")
```

which will return 1 if wall sensor does detects a wall, and 0 otherwise. For details about the meaning of the values returned, see the iRobot Create Open Interface Manual (iRobot Inc., 2006a).

There are several ways to get the sensor values using Java. The most direct is to use the `short iRobotCreate.sensorCache[]` attribute. The latest sensor readings are in the vector. To find the indici corresponding to a particular sensor, index by the constant `iRobotCreate.iRobotCommands.S_`*sensor-name* where *sensor-name* is the name of the sensor given in §4. For example, to get the value of the wall sensor as in the above Lisp example, use:

```
short wall = sensorCache[S_Wall];
```

Of course, this only works when the code is inside a subclass of `iRobotCreate` as this `sensorCache[]` is protected.

4.2 Reacting to Sensor Value Changes

To execute code (within an agent class) when any of the values of these variables changes, use the following code:

```
1  BeliefObserver bo = casa.jade.BeliefObserver.onValueChange(agent.kBase, "x", null,
2    new Runnable1<Object, Object>(){
3      @Override
4      public Object run(Object theValue) {
5        <code>
6        return null;
7      }});
```

In this code, line 1 invokes the `onValueChange()` method, specifying the agent's belief base[1], the name of the variable to monitor, any filter, and (on the following lines) the code packed in a `Runnable1<Object, Object>` object. Lines 2-7 instantiate an anonymous class that serves to contain the code ("<code>") that will be executed every time the value of the variable "x" changes. The parameter value of `run()` will be new value of the variable "x" (as a Long, Double, or String).

To stop the observer when you no longer want to execute the code upon sensor changes, us the following code:

```
agent.kBase.removeObserver(bo);
```

This serves to remove the observer from the KB and stop further invocations of the code in the anonymous class.

[1] also referred to as "knowledge base".

Bibliography

Common-Lisp.net, 2011. *Armed Bear Common Lisp (ABCL) - Common Lisp on the JVM*. URL `http://common-lisp.net/project/armedbear/`.

Element Products Inc., 2007. *BAM for the iRobot Create*. Element Products Inc., Broomfield, Colorado. URL `http://www.elementdirect.com/files/10542B.pdf`.

iRobot Inc., 2006a. *iRobot Create Open Interface*. iRobot Inc., Bedford, Mass. URL `http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf`.

iRobot Inc., 2006b. *iRobot Create Owner's Manual*. iRobot Inc., Bedford, Mass. URL `http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Manual_Final.pdf`.

Kremer, Rob, 2012. *CASA User Manual*. Technical report, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada. URL `http://casa.cpsc.ucalgary.ca/doc/CasaUserManual.pdf`.